Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 52 (2015) 538 – 545

Procedia Computer Science

The 6th International Conference on Ambient Systems, Networks and Technologies (ANT 2015)

Modelling multi-agent systems with category theory

Olga Ormandjieva*, Jamal Bentahar, Jinzi Huang, Heng Kuang

Concordia University, 1455 de Maisonneuve West, Montreal H3G 1M8, Canada

Abstract

The aim of this paper is to formalize the connection between two widely separated branches of knowledge: multi-agent systems (MAS) and category theory. The relationship of category theory to multi-agent systems is as follows: (1) agents and their relations are represented as categorical concepts; and (2) verification of system properties becomes constructive proof in category theory. Proposing a categorical approach to specify MAS properties requires a deep understanding of both the system and these properties in order to be able to abstract and reason about them in a categorical framework. The paper uses the MAS fault-tolerance property as an application of the categorical proof.

Peer-review under responsibility of the Conference Program Chairs

Keywords: Category theory; multi-agent systems; formal methods.

1. Introduction

The rapidly growing complexity of integrating and monitoring the computing multi-agent systems (MAS) is beyond the capabilities of even the most expert system and software developers[1,2]. The word “complexity” in this paper refers to the structural and behavioral complexity of a MAS. The most familiar and effective way of dealing with complexity is via formal methods, using mathematical techniques such as abstract algebra, logic and discrete mathematics for representing and reasoning about information required to build software systems. The goal of this paper is to tackle MAS complexity by providing a foundation for a graphical formalization of MAS requirements

* Corresponding author. Tel.: +1 514 848-2424 (7810); fax: +1 514 848-2830.
E-mail address: olga.ormandjieva@concordia.ca

Peer-review under responsibility of the Conference Program Chairs
doi:10.1016/j.procs.2015.05.031

(both functional and non-functional) in terms of agents' interrelations and communication properties. It is thus important to point out that visual modeling methodologies without a mathematical basis could not qualify as a formal method. Category Theory (CT) is our choice of a formalism, which offers techniques for manipulating and reasoning about diagrams for building the hierarchies of system complexity, and allows systems to be used as components of more complex systems[3]. The importance of the CT in software engineering lies in its ability to formalize the notion that things that differ in substance can have an underlying similarity of "structural" form. Compared to other software concept formalizations, CT is not a semantic domain for formalizing the description of components or their connectors, but expresses the semantics of the components' communication structure (interconnection, configuration, instantiation, and composition). Moreover, CT is an area of study that examines in an abstract way the properties of systems by formalizing them as collections of objects and arrows (morphisms). Hence, using CT allows us to concentrate on the morphisms or relationships among objects, instead of objects' representations. This is suitable for agent-based systems, since the most important concept among agents is their communication.

The long-term aim of MASs research is to develop mechanisms, which enable agents to interact with each other and understand their interactions. Such a goal raises a number of challenging issues, which are wholly centered on the research problem of how agents interact with each other to successfully satisfy their design goals. We propose a CT approach to model MAS with the purpose of exploring answers for the following research questions:

RQ1. *How can each agent be modelled with CT? More specifically: What are the components of each agent? How do we model each component with CT? How do we model the relations among the components within an agent?*

RQ2. *How do we formally model the communication structure of agents in MAS with CT? What are objects and morphisms to be used to capture the transformation from MAS to its categorical model? Since agents and their communication can be classified into different types, how do we model these types with CT?*

RQ3. *How can CT be used to model and verify MAS properties such as fault-tolerance? How do we make sure that agent communications are correctly specified?*

We present a novel approach for the MAS properties' verification. Our approach constructs a model of the MAS and the agent communication in category theory and checks the system properties by constructing categories and functors. For instance, failing to construct a functor between a MAS category and a category specifying its desired communication properties would indicate a potential modelling error. In this paper a proof of concept is provided by proposing CT models of MAS and illustrating the verification by construction on a small example. The proofs in category theory are constructive and therefore could be translated into algorithms and automated[4]. The scope of this paper excludes the automation of those CT proofs.

The rest of this paper is organized as follows. In Section 2, we introduce the basic category theory concepts. Sections 3 and 4 are the core of this paper that present the modeling of MAS by using CT concepts. Section 5 defines the fault-tolerance property and explains its verification by construction. The proposed MAS-CT methodology is summarized in section 6. Section 7 surveys the related work. The paper is concluded with future work directions in section 8.

2. Some Category Theory Definitions and Notations

CT is an advanced mathematical formalism that is independent of any modeling or programming paradigm, which is adequately capable of addressing "structure" [5,6,7]. In mathematics, CT is an abstract way of agreement between various mathematical structures and relationships among them. For example, sets are abstracted as objects and functions between sets are abstracted as morphisms. So a category named SET will have objects as sets and morphisms as all functions between them. A *category* C consists of the following components: i) *Objects:* A, B, C, etc. denoted ob(C), ii) *Morphisms:* f, g, h, etc. (hom(C)) where for each arrow f there are given objects: dom (f), cod (f) called the domain and codomain of f. We write: f: A → B to indicate that A = dom(f) and B = cod(f). iii) *Composition*: Given categories A, B, C, D and arrows f: A → B and g: B → C, i.e. with: cod(f) = dom(g), there is a given arrow: g ∘ f: A → C, called the composite of f and g; iv) *Identity:* For each object A there is a given arrow id_A: A → A, called the identity arrow of A; v) *Associativity***:** h ∘ (g ∘ f) = (h ∘ g) ∘ f, for all f: A → B, g: B → C, h : C → D; vi) *Unit:* f ∘ id_A = f = id_B ∘ f, for all f: A → B. Please note that in this paper, the identity morphisms are not displayed in most figures. A *functor* F: C → D between categories C and D is a structure preserving mapping of

objects to objects along with morphisms to morphisms in the way of: 1) F (f: A → B) = F (f): F (A) → F (B); 2) F (g ◦ f) = F (g) ◦ F(f); and 3) F (id_A) = $id_{F(A)}$. Key notions for the theoretical background of this paper are *initial* and *terminal* objects in a category. An *initial object,* where one exists in C, is an object for which every other object of C is a codomain of a unique morphism with the initial object as a domain. A *terminal object* has every object of C as the domain of a unique morphism where the terminal object is a codomain. More formally, in any category C an object is called *initial object* I if for any other object X in C, there is a unique morphism I→X. An object is called *terminal object* T if for any other object X in C, there is unique morphism X→T. A *discrete category* is a category where the morphisms are only identity morphisms. For example, suppose X and Y are different objects in category C, morphism from X to X only can be X's identity morphism, and morphism from X to Y will not exist. More formally, i) mor (X, X) = {id_X} for all objects X, and ii) mor (X, Y)= ∅ for all objects X≠ Y. A *subcategory* S of a category C is given by a subcollection of objects of C, denoted ob(S), a subcollection of morphisms of C, denoted hom(S). such that for every X in ob(S), the identity morphism id_X is in hom(S), for every morphism f : X → Y in hom(S), both the source X and the target Y are in ob(S), for every pair of morphisms f and g in hom(S) the composite g ◦ f is in hom(S) whenever it is defined. These conditions ensure that S is a category in its own right: the collection of objects is ob(S), the collection of morphisms is hom(S), and the identities and composition are as in C.

In the following sections we introduce categorical modelling of multi-agent systems. We zoom into agent's structure, and represent the main concepts: plans, goals, beliefs, and their relationships via CT. At the end we will zoom out to the level of entire multi-agent system, and represent MAS by using CT constructs.

3. Agent in CT Representation

MAS can be defined as "systems in which several interacting, intelligent agents pursue some set of goals or perform some set of tasks" [8]. An agent is a computer system that is situated in an environment, and designed to perform autonomous actions in this environment in order to meet its objectives. This paper explores a particular type of agents, namely, Belief-Desire-Intention (BDI) agents. Abstract BDI agents have been investigated by many researchers from both theoretical specification perspective and practical design perspective. BDI modal logic that extends temporal logics with BDI modalities has been widely used in MASs. However, the semantics of this logic based on the concept of possible words and accessibility relations is not computationally grounded and the accessibility relations of the three modalities are defined in a very abstract way that does not captures the intuitive differences between them. Nevertheless, there is a gap between modelling agent communication infrastructure and the representation of internal agent concepts. This section presents an approach to bridge this gap by developing an expressive categorical model that precisely captures the BDI aspects of an agent and agent communications with a level of abstraction that is suitable for precise formal analysis and computation. The idealized architectural components of a BDI system in our work are as follows: i) beliefs that represent the informational state of the agent, including a BeliefSet - a storage of agent's beliefs; ii) Desires (objectives the agent would like to accomplish), including Goals that are the desires that has been adopted for active pursuit by the agent, iii) Intentions summarized in Plans (that is, the desires the agent has begun executing), and iv) Plans (sequences of actions that an agent can perform to achieve one or more of its intentions). Table I summarizes the corresponding categories.

Table I. BDI Agent Categories: Definitions

FactSet	*A discrete category where objects are "facts" and the only morphisms are identity morphisms. The facts are information or knowledge about the agent's environments and system.*
BELIEF	*A category of Sets whose objects are the categories FactSets (one FactSetBase and one FactSetNull are included as default), and morphisms are "subset _of".*
Action	*A discrete category whose objects are "actions" denoted by Act1, Act2, … and the only morphisms are identity morphisms*
ObjectNull	*ObjectNull and its identity morphism are useful for catching "non-useful" or "non-related" objects and morphisms from other categories through defined functor (relation).*
Plan	*A category whose objects are "actions" denoted by Act1, Act2, … and the morphisms model a partial order relation "before"*
PLAN	*A category whose objects Plan1, Plan2… are plans and the morphisms are "before". PLAN has a null object PlanNull.*
GOAL	*A category whose objects are the agent's goals and the morphisms model "higher _priority".* *The definition of "higher _priority" is: the domain of this morphism has higher or the same priority level than the co-domain.* *GoalNull stands for an empty object with no morphism from or to other goals.*

Type Category	*A category whose objects represent the object types denoted by ObjType(TypeCategory), and whose morphisms represent the morphism types denoted by MorType(TypeCategory).*
Priority	*A category whose objects are "1", "0", "-1" as well as "unsigned", and morphisms are "bigger or equal" denoted by "≥". Integer "1" stands for high priority, "0" stands for median priority, "-1" stands for low priority, and the "unsigned" means the priority is unknown that doesn't have any relations (morphisms) with other objects.*

Plans represent the agent's means to act on the requests initiated by other agents or from its environment, and one single plan is abstracted as a sequence of actions. Therefore, plans of an agent are collections of sequences of actions, which are performed in a discrete time. A single plan is abstracted as a sequence of actions in Plan category. Here, "actions" are defined as an abstraction of agents' reaction to the environment events and morphisms are named "before" [9,10]. The morphism "before" models the partial order between the actions. The first action of the sequence named trigger action represents the action of receiving a "trigger event message". The received messages can come from either internal or external source. Internal messages are those sent from the owner of the plan, and external messages are those sent from other agents or the environment. So when we say a plan is started, we mean the trigger action of this plan has been performed. Plan has a null object, denoted as Act_{Null} ; it is used to catch exceptions. Let f: $Act_1 \rightarrow Act_2$ and g: $Act_2 \rightarrow Act_3$ be morphisms of type "before". The composition of *f* and *g* (g ◦ f): $Act_1 \rightarrow Act_3$, of type *before* is meaningful: Act_1 is performed earlier than Act_3. Plan category satisfies associativity and unit laws. Therefore, the validity of the category Plan is proved. A category of all plans of one agent is named PLAN. PLAN also has a null object $Plan_{Null}$. The following definitions abstract the relations between categories: *Action*, *Plan*, and *PLAN* as functors, see Table II.

Table II. BDI Agent Functors: Definitions

sequence_action: *Action→ Plan*	*Maps all the "actions" of Action to "actions" of Plan, and all the identity morphisms of Action to the corresponding identity morphisms of Plan's objects.*
refined _by _plan: *Plan → PLAN*	*The functor "refined _by _plan" maps all the "actions" of Plan to "plans" of PLAN, and all the morphisms (include identity) of Plan to identity morphisms of PLAN.*
timing_action: *Plan → Discrete-Time*	*Maps objects (actions) of Plan to objects (time unit expressed as integers) of Discrete-Time, and maps morphisms of Plan (before) to morphisms "(<)" of Discrete-Time.*
timing_plan: *PLAN → Discrete-Time*	*Maps objects (plans) of PLAN to objects (time unit expressed as integers) of Discrete-Time, and maps morphisms of PLAN (before) to morphisms "(<)" of Discrete-Time.*
self_PLAN: *PLAN → PLAN*	*Identity functor of the category PLANmodels agent's ability to update its plans, and provides a way for PLAN to remove its un-achievable plans.*
plan_goal: *PLAN → GOAL*	*Every object (Plan) in PLAN is mapped to one object (Goal) in GOAL, and the morphisms "before" in PLAN are mapped to the corresponding morphisms "higher _priority" in GOAL.*
plan_belief:*PLAN→BELIEF*	*The functor "plan _belief" means agent plans have access to read or write facts from agent's BELIEF.*
goal_belief: *GOAL → BELIEF*	*Every object "Goal" in GOAL will be mapped to an object "FactSet" in BELIEF, and the morphisms "higher _priority" in GOAL will be mapped to identity morphism of FactSetNull in BELIEF.*
assigned_priority: *GOAL → Priority*	*Models the fact that the objects (goals) in GOAL can be assigned to corresponding priority levels in category Priority. The morphisms (higher _priority) in GOAL are mapped to morphisms ("≥") .*

The category modelling in this section captures some important properties of multi-agent systems such as *action sequentiality.* Their sequential order relations that are captured by the morphism "*before*" in Plan can be mapped into the time objects and their relations "<" in Discrete-Time category by a functor *timing _action*. All object (actions) of Plan are mapped to plans in the category PLAN and all the morphisms (including the identity) of Plan are mapped to identity morphisms of PLAN by the functor *refined _by _plan.*

The sequential order of plans captured by the morphism "*before*" in PLAN can be mapped into time objects and their relations "<" in Discrete-Time by a functor *timing _plan*. With this property, we are able to prove that a plan starts at the same time that its *triggering action* is performed, and all the following actions of this plan occur later in time. The PLAN identity functor *self _PLAN* gives an agent the ability to update its plans, and provides a way for *PLAN* to remove its un-achievable plans: the achievable plans will be mapped to plans in agent's updated *PLAN'* and the non-achievable ones will be mapped to the exception catcher, the $Plan_{Null}$ in *PLAN'* (see Fig. 1).

Discrete Time category is extremely useful to guarantee that *actions* of Plan and *plans* of PLAN occur in a correct time order by mapping them to Discrete Time with the functors *timing_action* and *timing_plan* accordingly, That is, if there is morphism *before:* $plan_A \rightarrow plan_B$ then the triggering actions Act_1 and Act_4 of the plans $plan_A$ and $plan_B$ are mapped by *timing_action* to time objects ordered by the morphism "<" : *timing_action* (Act_1) < *timing_action* (Act_4) where *timing_plan* ($plan_A$)= *timing_action*(Act_1) and *timing_plan* ($plan_B$)= *timing_action*(Act_4) (see Fig. 1).

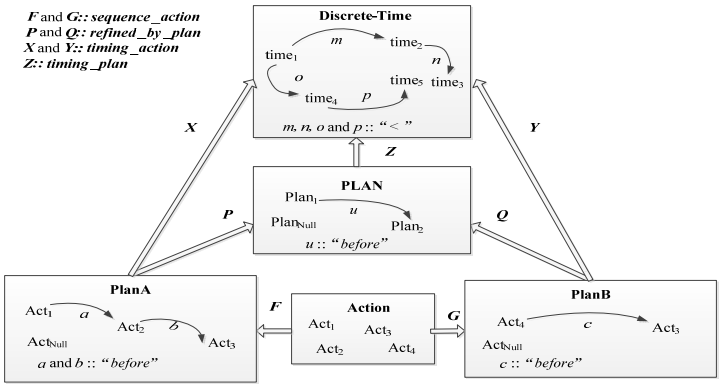

Fig. 1. Modeling agent plans with CT

Goals make up agent's motivational stance and are the driving forces for its actions. Therefore, the representation and handing of goals is one of the main features of agents. In fact, each agent has a set of goals that are dispatched by plans. Goals with higher priority need to be performed earlier than goals with lower priority. *Priority* category is used to set up the order of importance or urgency of different goals. Goals can be classified into different levels of priority by using the functor *assigned_priority*.

Beliefs represent agent's knowledge or information about the environment and itself. Beliefs are built from different information chunks about the agent's environments and the system called *facts*, which are organized into *fact sets*. Based on different usage, facts are classified into *FactSet* categories. Two special FactSets categories need to be introduced: $FactSet_{Base}$ and $FactSet_{Null}$. $FactSet_{Base}$ includes all the facts that every other FactSet has, and $FactSet_{Null}$ contains no facts at all or it's an empty set. Inside each FactSet (including the $FactSet_{Base}$, except for $FactSet_{Null}$) we define a special object denoted as $Fact_{Null}$. $Fact_{Null}$ is a null fact, which doesn't have any morphisms (except for his identity). The BELIEF represents an inheritance structure, see Fig. 2 (a). This structure guarantees *data's consistence* because one agent has only one $FactSet_{Base}$, and all other fact sets are subset of $FactSet_{Base}$. In BELIEF, any FactSet is a subset of $FactSet_{Base}$, and more formally, every fact in FactSet can be found in $FactSet_{Base}$. Similarly, $FactSet_{Null}$ has the "subset _of" relations to every FactSet. Using the definitions of initial and terminal objects, NullSet is the initial object and BaseSet is the terminal object in BELIEF.

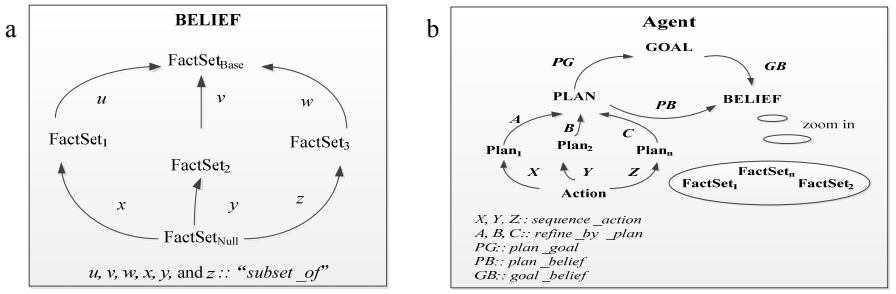

Fig. 2. (a) Representation of the BELIEF category; (b) Categorical Representation of an Agent

In the BDI systems each plan must dispatch a goal, but the goal can be a null object. Basically, in an agent, the plans have to dispatch relevant goals. The functor *plan _goal* PG: PLAN → GOAL captures the fact that every plan from PLAN category dispatches a goal from GOAL category. The PG functor also guarantees that a plan can only dispatch one corresponding goal, and different plans can dispatch the same goal. The functor *plan _belief* PB: PLAN → BELIEF means agent plans have access to read or write facts from agent's BELIEF. By using this category, the property "one plan can only access one *factset*" can be captured. The functor *goal _belief* GB: GOAL → BELIEF ensures that every goal has an access to read facts or knowledge from agent beliefs.

PLAN, GOAL, BELIEF and FactSet have *self** (identity) functors, which allow each category to have ability to remove their objects such as unreached goals (in *GOAL)* or unachievable plans (in *PLAN)* by mapping them to

ObjectNull. There is no difference between *ObjectNull* and other categories' objects, except that it doesn't have any real meaning or content, and it doesn't have any relationship with other objects. To sum up, a BDI agent can be represented formally by the categories Action, Plan, PLAN, GOAL, BELIEF, FactSet, and the functors*: plan _goal*, *plan _belief*, *goal _belief*, *refined _by _plan* and *sequence _action* (see Fig. 2 (b)).

In our work, an object Obj of a type category represents an abstraction of a real-world entity type, the set of input events activating that type of the entity modelled with the morphisms whose codomain is Obj, and the corresponding output events captured as morphisms whose domain is the Obj. As an example of using TypeCategory we consider MyCategory whose objects are denoted by Obj(MyCategory) and the morphisms denoted by Mor(MyCategory). A functor F constructed from MyCategory to TypeCategory will map each object of MyCategory to a type (an object of TypeCategory), and map each morphism of MyCategory to a type (a morphism of TypeCategory), so that F(Obj(MyCategory)) = ObjType(TypeCategory) and F(Mor(MyCategory)) = MorType(TypeCategory). Constructing successfully the functor F will verify MyCategory against the MAS communication structure's specification captured in TypeCategory.

4. Modeling MAS with CT

MAS is a category that models the agents communication structure, whose objects are agents and the morphisms capture the communication between those agents. Using this category, we are able to model all agents' relationships in a system, such as which agents have the ability to communicate directly, which agents need other agents to delegate messages to, and which type of communication is taking place between the agents.

In our approach to formalizing multi agent systems, there exists one special agent: *Repository Type* agent. Repository Type is a type category whose objects are categories that represent the types of agents, and whose morphisms represent the types of communication between agents. In other words, Repository Type represents the semantics of communication while MAS category captures their structure.

From MAS to Repository Type, there exists a functor *MAS-rep* that describes the relations between MAS and Repository Type objects. *MAS-rep* provides mapping rules to map objects and morphisms within Agent A, Agent B and Agent C's Action, Plan, PLAN, GOAL and BELIEF to objects and morphisms within Type X and Type Y's ActionType, PlanType, PLANType, GOALType and BELIEFType (see Fig. 3). In other words, constructing successfully the MAS-rep will verify the MAS structure against the specified type of communication.

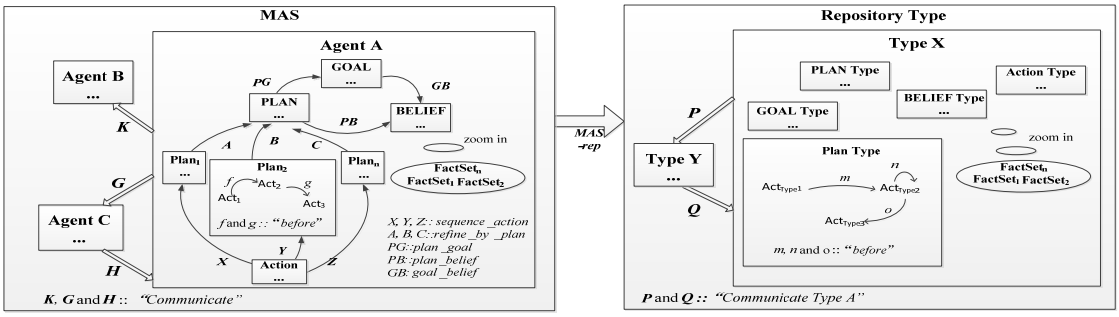

Fig. 3. Mapping MAS to Repository Type

Coming up with an accurate MAS model using the same CT-based formal modelling approach is very valuable not only because it is required as a basis to precisely state MAS properties and perform formal verification by construction, but also because it summarizes and condenses important agent and MAS aspects visually in several CT constructs.

5. Modeling and verifying MAS fault-tolerance with CT

In our approach, MAS properties are established through constructive proofs, that is, a proof from the verification that the construction satisfies its specification. The approach we undertake relies on the fact that category theory is constructive: proofs of existence are explicit constructions using the representations of MAS categories and functors

presented in the previous sections.

We now illustrate the constructive proof approach on the fault-tolerance property. Fault-tolerance is defined as a property that enables a MAS to continue operating properly in the event of the failure of (or one or more faults within) some of its components. The chosen MAS fault-tolerance strategy is a replacement of a damaged agent with another agent enabled to complete the tasks of the damaged agent, that is, the actions, plans, goals and beliefs of the replacement agent have to include those of the damaged agent. To verify if a damaged agent A can be replaced by an agent R, the system will first verify whether or not both A and R are of the same type by constructing a functor *MAS-rep* (see Fig. 3). Next, if both A and R belong to the same type (that is, functor *MAS-rep* maps both A and B to the same agent type), then the system will construct *embedding functor* EF: A → R where the image of EF is a subcategory S of R and EF is necessarily injective on objects up-to-isomorphism. More specifically, it will map each one of A's categories as follows: EF_{Action} : $Action_A \rightarrow Action_S$, EF_{Plan} : $Plan_A \rightarrow Plan_S$, EF_{PLAN} : $PLAN_A \rightarrow PLAN_S$, EF_{GOAL} : $GOAL_A \rightarrow GOAL_S$, $EF_{FactSet}$: $FactSet_A \rightarrow FactSet_S$, and EF_{BELIEF} : $BELIEF_A \rightarrow BELIEF_S$. Partial illustration of the embedding functor is shown in Fig. 4 where $BELIEF_S$ consists of $FactSets_{1\_A, 1\_B, Null, Base}$ and the morphisms u_2, v_2, x_2, y_2 and z_2:

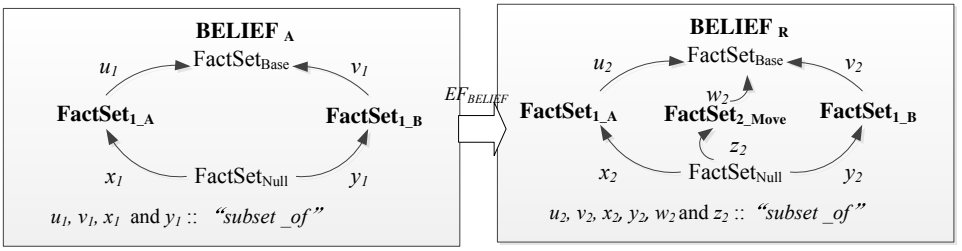

Fig. 4. Illustration of an embedding functor EF_{BELIEF}

Failing to construct even one of the above functors would prevent the subsystem from using the agent R as a replacement for the agent A, meaning that the fault-tolerance property cannot be realized.

6. MAS-CT Methodology

The methodology proposed here stems from three basic components: i) a precise notion of an agent captured as a CT category (RE: RQ1), ii) a MAS of communicating agents and their communication composition modeled as categories and functors, including the agent/MAS incremental changes by self* (identity) morphisms (RE: RQ2), and iii) MAS properties' verification by construction (RE: RQ3). Verification by construction involves a construction of a category, a definition of the properties to be satisfied by that category that are modeled as another category, and a construction of a functor that checks the satisfaction of those properties. Correctness is verified only when such a functor can be constructed. The main steps of the MAS-CT methodology are summarized in Fig. 5.

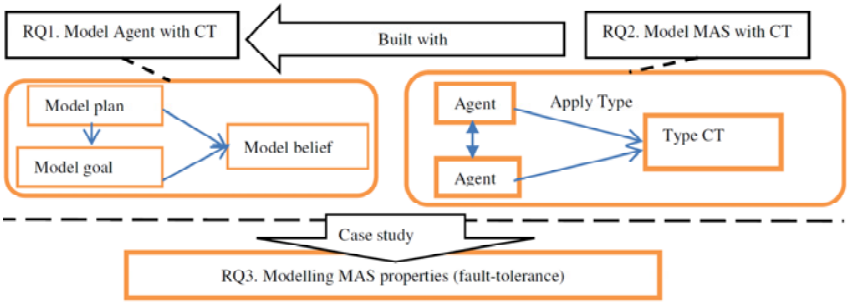

Fig.5. MAS-CT Methodology

7. Related Work

The only published work that we found on modelling complex systems using CT served as the foundation for our research[11]. The author defined a specification language of autonomous systems based on CT, but no systematic

methodology and specific self-* properties, such as self-healing or fault-tolerance, were proposed in that publication. More recently, CT was applied to model MASs, and a categorical model of MASs was stated in [12,13] as the MAS category. In [14] the authors focused on a construction of a transformation system for MAS based on categorical notions and previously introduced MAS category. There is also some related work regarding the formal specification of intelligent swarm systems such as our case study Mars-world. Authors in [15] illustrate a formal task-scheduling approach and model the self-scheduling behaviour for Prospecting Asteroid Mission with an Autonomic System Specification Language.

Our research builds on the related work listed above, since our goal is to propose a systematic and formal methodology based on CT to model and specify the MAS properties, which could be applied to model not only MAS, but also service-oriented systems and ambient systems.

8. Conclusions and Future Work

MAS have been widely proposed and applied to various application domains, such as space exploration missions. MAS can offer greater redundancy, efficiency, and scalability; however, they also raise new challenges, for instance, complex and often unexpected group behavior, which require a formal specification as well as verification. This paper proposed a formal modelling of MAS with CT, focusing on the morphisms or relationships between objects i.e., as agents, rather than concentrating on those agents' representations.

The MAS design of intelligent agents modelled and verified with CT can be realized further in Jadex, a high-level Java agent development framework based on the BDI agent architecture[16] . Eventually, a source code template can be automatically generated according to the MAS model by using a model transformation tool proposed in [16] . One of few MAS modeling challenges is specification and verification of the emergent behavior. This could be a possible path for future research.

Acknowledgements

We would like to thank Dr. Peter Grogono for his helpful comments on the manuscript.

References

1. Ferber J. *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison-Wesley; 1999.
2. Wooldridge M. *An Introduction to Multiagent Systems*. John Wiley & Sons, Inc., New York, NY; 2009.
3. Awodey S. *Category Theory*. Oxford University Press; 2006.
4. Rydeheard DE, Burstall RM. *Computational category theory*. Prentice Hall Englewood Cliffs; 1988.
5. Barr M, Wells C. *Category Theory for Computing Science*. Prentice-Hall, Englewood Cliffs, NJ; 1990.
6. Whitmire S. *Object Oriented Design Measurement*. John Wiley & Sons, Inc., New York, NY; 1997.
7. Fiadeiro J. *Categories for Software Engineering*. Springer Verlag, Berlin Heidelberg; 2005.
8. Weiss G. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press; 1999.
9. Kuang, H., & Ormandjieva, O. Self-Monitoring of Non-functional Requirements in Reactive Autonomic Systems Framework: A Multi-agent Systems Approach. In: *The Third International Multi-Conference on Computing in the Global Information Technology ICCGI'08;* 2008. p. 186-192. IEEE.
10. Kuang H, Ormandjieva O, Klasa S, Khurshid N, Bentahar J. Towards specifying reactive autonomic systems with a categorical approach: a case study. *Studies in Computational Intelligence. Springer Berlin Heidelberg* 2009; 253: 119-134.
11. Lee. WM Modelling and Specification of Autonomous Systems using Category Theory. PhD Thesis, University College of London, London, UK, October 1989.
12. Pfalzgraf J. On Categorical and Logical Modelling in Multiagent Systems. *Anticipative and Predictive Models in Systems Science* 2005; 1: 93-98.
13. Pfalzgraf J. Soboll T. On a General Notion of Transformation for Multiagent Systems. In: *Proceedings of the 10th World Conference on Integrated Design & Process Techonology.* Antalya, Turkey; 2007.
14. Kuang H, Jamal Bentahar J, Ormandjieva O, Shafieidizaji N, Klasa S. Formal Specification of Substitutability Property for Fault-Tolerance in Reactive Autonomic Systems. In: *Proceedings of SoMeT 2010:* p. 357-380.
15. Ormandjieva O, Vassev E. Towards ASSL Specification of Self-Scheduling Design and Monitoring in Team-Robotics Modelled with AS-TRM. Novel Algorithms and Techniques In: *Telecommunications, Automation and Industrial Electronics*. Springer; 2008. p. 68-76.
16. Shafiel-Dizaji N. Multi-Agent Approach to Modelling and Implementing Fault-Tolerance in Reactive Autonomic Systems. *Master Thesis, Concordia University*; 2011.